Decision Trees in R

Therese Donovan

2020-03-24

Overview

This script is a translation of the decision tree. In this module, you enter into the "Microsoft Exhibit" at your state fair, where a gatekeeper gives you 100 betting tickets. The game is that you flip a coin, and if it comes up heads you win you earn money, and if it comes up tails you lose money. There are two tents, and you must **decide** which tent to play in:

- Red booth: In the red booth, if the coin flip is heads, you win $10. If the coin flip is tails, you lose $1.
- Blue booth: In the blue booth, if the coin flip is heads, you win $100. If the coin flip is tails, you lose $20.

Which booth will you enter?

The Booth Decision Tree ——

Set up the payouts.

Step 1 is to set up our outcomes. These are the outcomes that are associated with each tip in a decision tree branch. Here, our outcomes are in dollars.

```
red.head <- 10
red.tail <- -1

blue.head <- 100
blue.tail <- -20
```

Sometimes you don't have payoffs that are monetary. You can "generalize" payouts as utility scores – what benefit will you receive given some outcome. These are general in that they are not necessarily monetary. And they are flexible in that each decision maker can assign their own, unique utility scores to outcome. So the same tree structure can have different tip outcomes (utility) for a decision maker that is risk-seeking versus someone who is risk-neutral versus someone who is risk adverse.

Probabilities

So, we now imagine a tree that starts with a decision. Red booth or blue booth? Once you enter a booth, you hand over your tickets, and each time a coin is flipped. Which path you follow in the tree established the "branch" you follow. The coin flip has an uncertain result; this juncture in the tree is called a "chance node" because we are unsure of the result. We need to establish the outcomes associated with the chance node (heads or tails) and the probability of observing each outcome. These must sum to 1. Here, we'll assume the coin is fair, so the probability of heads is 0.5 and the probability of tails is 0.5.

```
heads <- 0.5
tails <- 1 - heads
```

Now, our tree is complete. We start with a decision: red or blue tent? For each alternative, there is a chance node (heads or tails). That means there are four branches in total in our tree. Each branch tip has an outcome associated with it (the payout). So now we can "roll back" the tree to the decision and calculate the expected value of each decision.

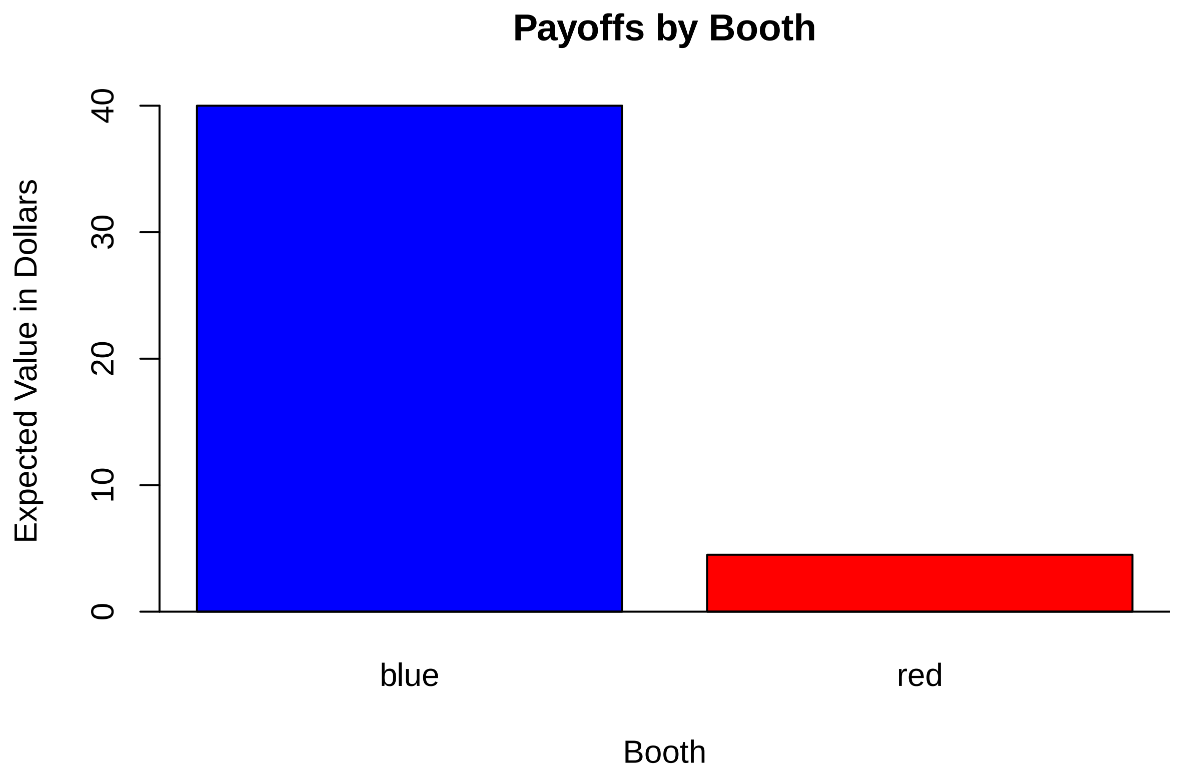
Expected Value for each Decision

```
EV.blue <- (heads * blue.head)  + (tails * blue.tail)
EV.red <-  (heads * red.head) + (tails * red.tail)
```

Let's make a barplot of these values

```
barplot(height = c(EV.blue, EV.red),
        col = c("blue","red"),
        names.arg = c("blue", "red"),
        main = "Payoffs by Booth",
        xlab = "Booth",
        ylab = "Expected Value in Dollars")

# add horizontal base
abline(h = 0)
```



Simulation

In class, Tony mentioned that some people may not be convinced that the expected value calculations really work. Can it really be that simple? He had us do a Monte Carlo experiment where we actually played our 100 bets 1 at a time. We then compared our Monte Carlo results to the expected value results.

Let's do that next. Here, we'll make a new function called `playBooth()`. This will include the inputs for the EV calculations, as well as 4 additional arguments that we need.

1. trials - how many tickets can you play? The default will be 100.
2. random.seed - used for reproducibility. If there is just one simulation, then by using the same random number seed, you can reproduce your results even in light of randomness.
3. sims - the number of simulations to run. The default will be 1 simulation, which consists of a certain number of trials.
4. bribe (for later) - the entrance fee to play in the blue booth. The default is 0.

All of the `playBooth()` function arguments will have default values that match, more or less, our spreadsheet version. Take a look at these now.

```r
# create the function
playBooth <- function(heads = 0.5,
                      red.head = 10,
                      red.tail = -1,
                      blue.head = 100,
                      blue.tail = -20,
                      trials = 100,
                      random.seed = 34,
                      sims = 1,
                      bribe = 0) {

  # set up tails
  tails <- 1 - heads

  # set the random number seed (for reproducibility)
  set.seed(random.seed)

  # set up a vector of trials
  trial <- 1:trials

  # set up empty dataframe to store results
  results <- data.frame()

  # now we begin our simulation loop ---------------

  # set up sim loop
  for (sim in 1:sims) {

  # set up vector of random numbers
  random.number <- runif(n = trials, min = 0, max = 1)

  # set up result
  result <- ifelse(random.number <= heads, yes = "heads", no = "tails")

  # set up the red tent payout
  red.payoff <- ifelse(result == "heads", yes = red.head, no = red.tail)

  # set up the blue tent payout
  blue.payoff <- ifelse(result == "heads",
                        yes = blue.head,
                        no = blue.tail)

  # cumulate the payoffs, and record the final cumulative value
  red.cumulative <- sum(red.payoff)
  blue.cumulative <- sum(blue.payoff)
  # take off the bribe to play in the blue tent
  blue.cumulative <- blue.cumulative - bribe

  # compile the simulation result
  sim.result <- data.frame(
      "Simulation" = sim,
      "Red" = red.cumulative,
      "Blue" = blue.cumulative)

  # bind the sim.result to our results dataframe
  results <- rbind(results, sim.result)

  } # end of sim

  # return the results of the function
  return(results)

} # end of function
```

To run our function, we just need to call it by name, and provide the arguments within the parentheses. Since all of the arguments have default values, we can use () to indicate that no arguments are passed (and the default values will be invoked).

```r
# run the function with all the default values
playBooth()
```

```
##   Simulation Red Blue
## 1          1 450 4000
```

Let's run it again. Notice that we get the exact same result.

```r
playBooth()
```

```
##   Simulation Red Blue
## 1          1 450 4000
```

Why did we get the same exact result, even though 100 random numbers are involved. The answer is that the same random.seed is was used in both runs (the default seed of 34).

If we want a new simulation, we must change the seed.

```r
playBooth(random.seed = 1004)
```

```
##   Simulation Red Blue
## 1          1 483 4360
```

Now we have run two simulations. We can run 1000 simulations by setting the sims argument to 1000. Let's store our outputs as a new R object called "monte.carlo".

```r
monte.carlo <- playBooth(sims = 1000)
```

As always, the `str()` function is terrific to look at the structure of any R object.

```r
str(monte.carlo)
```

```
## 'data.frame':    1000 obs. of  3 variables:
##  $ Simulation: int  1 2 3 4 5 6 7 8 9 10 ...
##  $ Red       : num  450 384 417 450 428 549 560 494 494 461 ...
##  $ Blue      : num  4000 3280 3640 4000 3760 5080 5200 4480 4480 4120 ...
```

Here, we can see that our object is a dataframe with 1000 rows and 3 columns. Let's have a look at the first 10 records:

```
head(monte.carlo, n = 10)
```

```
##    Simulation Red Blue
## 1           1 450 4000
## 2           2 384 3280
## 3           3 417 3640
## 4           4 450 4000
## 5           5 428 3760
## 6           6 549 5080
## 7           7 560 5200
## 8           8 494 4480
## 9           9 494 4480
## 10         10 461 4120
```

Remember, the numbers show the earnings in the red and blue booth after 100 trials (coin flips).

Summarize the outputs

There are many ways to summarize a dataframe in R. One function that provides a lot of information is the `quantile()` function. Let's use this for both the red and blue results so we can compare them.

```
quantile(monte.carlo$Red)
```

```
##   0%  25%  50%  75% 100% 
##  296  406  450  483  648
```

```
quantile(monte.carlo$Blue)
```

```
##   0%  25%  50%  75% 100% 
## 2320 3520 4000 4360 6160
```

Here, the 0% quantile shows the minimum earnings in each tent, and the 100% quantile shows the maximum earning in each tent. The 50% quantile is the median. Let's now compare these to our expected value calculations – the calculations done at the top of this document.

```
# EV.blue <- (heads * blue.head)  + (tails * blue.tail)
EV.blue * 1000
```

```
## [1] 40000
```

```
# EV.red <-  (heads * red.head) + (tails * red.tail)
EV.red * 1000
```

```
## [1] 4500
```